A New Low-Power Recoding Algorithm for Multiplierless Single/Multiple Constant Multiplication

A.K. Oudjida, M.L. Berrandjia
Microelectronics and Nanotechnology Division
Centre de Développement des Technologies Avancées
Algiers, Algeria
a_oudjida@cdta.dz

N. Chaillet
AS2M Department
FEMTO-ST Institute
Besançon, France
nicolas.chaillet@femto-st.fr

***Abstract*— Optimizing the number of additions in constant coefficient multiplication is conjectured to be a NP-hard problem. In this paper, we report a new heuristic requiring an average of 29.10% and 10.61% less additions than the standard canonical signed digit representation (CSD) and the double base number system (DBNS), respectively, for 64-bit coefficients. The maximum number of additions per coefficient is bounded by (*N*/4)+2, and the time-complexity of the recoding is linearly proportional to *N*, where *N* is the bit-size of the constant. These performances are achieved using a new redundant version of radix-2^8 recoding.**

Keywords—Double Base Number System (DBNS); High-Speed and Low-Power Design; Multiplierless Single/Mutiple Constant Multiplication (SCM/MCM); Radix-2^r Booth recoding.

I. BACKGROUND AND MOTIVATION

Many applications in DSP and control, such as linear time invariant (LTI) filters/controllers, involve the computation of a large number of multiplications of one variable by a set of constants. To be efficiently handled, the implementation must be multiplierless, that is, using exclusively additions, subtractions and shifts. This problem is known as single/multiple constant multiplication (SCM/MCM) and is conjectured to be NP-hard [1]. A big number of heuristics have been proposed. They are classified into four categories:

- Digit-recoding heuristics such as CSD [2] and DBNS [3] ;
- Common subexpression elimination (CSE) using pattern matching. Examples are Lefèvre [4] and Boullis [5];
- Directed acyclic graph (DAG) based algorithms such as Hcub [6], H(k) [7], and MAG [8];
- Mixed algorithms combining CSE and DAG such as the recent optimal algorithm BIGE [1].

A good survey and a detailed comparative study showing pros and cons of various algorithms is given in [1][6][9].

Despite the big number of proposed heuristics, the *vast majority* of LTI system optimizations use the CSD representation for constant encoding [10]. The rational is that:

- CSD recoding is *easy* to implement;
- The adder complexity in CSD is *known*, which is *not* the case for the other heuristics [1][11]. In CSD the number of adders is bounded by (*N*+1)/2-1 and tends asymptotically to an average value of (*N*/3)-8/9, which yields to 33% saving over the naïve add-and-shift approach.
- CSD requires a *linear* computational time, contrary to its counterparts that require an *excessive* runtime (Table I) and storage, which makes them impractical for high values of *N*, at least for the current compute power.

TABLE I. ASSYMPTOTIC RUNTIME SUMMARY

Algorithm			Runtime
Name	*Author*	*Year*	
BIGE	Thong [1]	2011	$O(2^N)$
H(k)	Dempster [7]	2004	$O(2^N)$
MAG	Gustafsson [8]	2002	$\Omega(2^N)$
–	Bernstein [12]	1986	$O(2^N)$
Hcub	Voronenko [6]	2007	$O(N^6)$
BHM	Dempster [13]	1995	$O(N^4)$
–	Lefèvre [4]	2001	$O(N^3)$
DBNS	Dimitrov [3]	2007	$O(N)$
CSD	Avizienis [2]	1961	$O(N)$

The central point of this work is the minimization of the total number of additions. Based on radix-2^r signed-digit number system [14] [15], a new *Redundant* Radix-2^r Recoding (R3) is proposed as an alternative to existing heuristics. Applied to the particular case of radix-2^8 with *N*=64, a saving of 29.10% is achieved over CSD, which yields to much less power consumption and more speed. In addition, the new recoding shows high aptitude for common subexpression elimination, which makes it a good candidate for MCM.

The paper is organized as follows. Section I outlines the necessity of a linear runtime heuristic with a high compression ratio to handle large bit-size constants. Section II introduces the new R3 algorithm, while Section III compares the results to CSD and DBNS recodings. Finally, Section IV provides some concluding remarks and suggestions for future work.

II. NEW REDUNDANT RADIX-2^r ALGORITHM (R3) FOR MULTIPLICATION BY A *N*-BIT CONSTANT

A *N*-bit *C* constant is expressed in radix-2^r as follows:

$$C = \sum_{j=0}^{(N/r)-1} \left(c_{rj-1} + 2^0 c_{rj} + 2^1 c_{rj+1} + 2^2 c_{rj+2} + \cdots + 2^{r-2} c_{rj+r-2} - 2^{r-1} c_{rj+r-1} \right) \times 2^{rj}$$

$$= \sum_{j=0}^{(N/r)-1} Q_j \times 2^{rj} \quad ; \qquad (1)$$

where $c_{-1} = 0$ and $r \in \mathrm{N}^*$. For simplicity purposes and without loss of generality, we assume that *r* is a divider of *N*.

This work is supported by "Centre de Développement des Technologies Avancées (CDTA), Algiers, Algeria, in collaboration with FEMTO-ST Institute, Besançon, France.

In eq. (1), the two's complement representation of C constant is split into N/r two's complement slices (Q_j), each of r+1 bit length. Each pair of two contiguous slices has one overlapping bit. To eq. (1), corresponds a digit-set $D(2^r)$ such as

$$Q_j \in D(2^r)=\{-2^{r-1},-2^{r-1}+1,\ldots,-1,0,1,\ldots,2^{r-1}-1,2^{r-1}\}.$$

The sign of Q_j term is given by c_{rj+r-1} bit, and $|Q_j|=2^{k_j}\times m_j$, with $k_j \in \{0,1,2,\ldots,r-1\}$ and $m_j \in OM(2^r)=\{1,3,5,\ldots,2^{r-1}-1\}\cup\{0\}$. $OM(2^r)$ represents the required set of odd-multiples in radix-2^r recoding, with $|OM(2^r)|=2^{r-2}$. Finally, C can be expressed as follows:

$$C=\sum_{j=0}^{(N/r)-1}(-1)^{c_{rj+r-1}}\times\left(m_j\times 2^{k_j}\right)\times 2^{rj}. \quad (2)$$

Equation (2) is *not redundant* since for each C constant corresponds a unique representation (m_j). To make the solution space larger in order to select a less adder-consuming representation of C, the recoding must be redundant. To achieve such a goal, we announce the following theorem:

Theorem 1. *In radix-2^r,* $|Q_j|=[A_j\times 2^p+(-1)^e\times B_j\times 2^h]$, where: $A_j,B_j\in\{0,1,3,5,\ldots,2^{(r/2)-1}-1\}$; $p\in\{0,1,2,\ldots,2^{r-1}-1\}$; $h\in\{0,1,2,\ldots,2^{(r/2)-1}-1\}$; and $e\in\{0,1\}$.

The proof of the above theorem is based on our Theorem (1) described in [16][17]. Note that different notations for $|Q_j|$ are possible. For instance: $37=1\times 2^5+5\times 2^0$ or $37=5\times 2^3-3\times 2^0$. We illustrate the idea for r=8, where $0\le|Qj|\le 128$. Equation (2) becomes:

$$C=\sum_{j=0}^{(N/8)-1}\left(A_j\times 2^p+(-1)^e\times B_j\times 2^h\right)\times(-1)^{c_{8j+7}}\times 2^{8j}$$

$$=\sum_{j=0}^{(N/8)-1}(Z_1+Z_2)_j\times(-1)^{c_{8j+7}}\times 2^{8j} \quad (3)$$

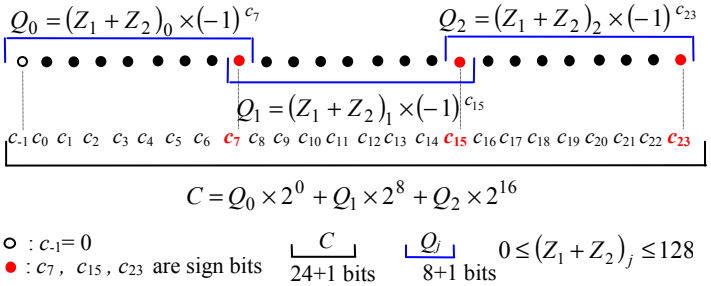
where $Z_1=A_j\times 2^p$; $Z_2=(-1)^e\times B_j\times 2^h$; $A_j,B_j\in\{0,1,3,5,7\}$; $p\in\{0,1,2,\ldots,7\}$; $h\in\{0,1,2,3\}$; and $e\in\{0,1\}$.

Note that $|Q_j|=(Z_1+Z_2)_j$. The partitioning of C constant according to eq. (3) is depicted in Fig. 1, while the recodings of odd and even $|Q_j|$ digits are separately denoted in Table II.

The product $C\times X$ becomes:

$$C\times X=\sum_{j=0}^{(N/8)-1}\left[\left(A_j\times X\right)\times 2^p+(-1)^e\times\left(B_j\times X\right)\times 2^h\right]\times(-1)^{c_{8j+7}}\times 2^{8j} \quad (4)$$

Note that when $A_j, B_j \in \{3,5,7\}$, one extra adder is needed since for instance: $3\times X=2\times X+X$.



$$C=Q_0\times 2^0+Q_1\times 2^8+Q_2\times 2^{16}$$

○ : c_{-1}= 0
● : c_7, c_{15}, c_{23} are sign bits

C: 24+1 bits; Q_j: 8+1 bits; $0\le(Z_1+Z_2)_j\le 128$

Figure 1. Partitioning of a 24-bit C constant using R3 algorithm.

TABLE II: ODD AND EVEN $|Q_j|$ DIGIT RECODING USING R3 ALGORITHM

Odd $\|Q_j\|$	$Z_1=A_j\times 2^p$	$Z_2=(-1)^e\times B_j\times 2^h$	$(Z_1+Z_2)_j$	Even$\|Q_j\|$	$(Z_1+Z_2)_j$
1	1×2^0	0×2^0	U_1	2	$2^1\times U_1$
3	3×2^0	0×2^0	U_3	4	$2^2\times U_1$
5	5×2^0	0×2^0	U_5	6	$2^1\times U_3$
7	7×2^0	0×2^0	U_7	8	$2^3\times U_1$
9	1×2^3	1×2^0	U_9	10	$2^1\times U_5$
11	3×2^2	-1×2^0	U_{11}	12	$2^2\times U_3$
13	3×2^2	1×2^0	U_{13}	14	$2^1\times U_7$
15	1×2^4	-1×2^0	U_{15}	16	$2^4\times U_1$
17	1×2^4	1×2^0	U_{17}	18	$2^1\times U_9$
19	5×2^2	-1×2^0	U_{19}	20	$2^2\times U_5$
21	5×2^2	1×2^0	U_{21}	22	$2^1\times U_{11}$
23	3×2^3	-1×2^0	U_{23}	24	$2^3\times U_3$
25	3×2^3	1×2^0	U_{25}	26	$2^1\times U_{13}$
27	7×2^2	-1×2^0	U_{27}	28	$2^2\times U_7$
29	7×2^2	1×2^0	U_{29}	30	$2^1\times U_{15}$
31	1×2^5	-1×2^0	U_{31}	32	$2^5\times U_1$
33	1×2^5	1×2^0	U_{33}	34	$2^1\times U_{17}$
35	1×2^5	3×2^0	U_{35}	36	$2^2\times U_9$
37	1×2^5	5×2^0	U_{37}	38	$2^1\times U_{19}$
39	5×2^3	-1×2^0	U_{39}	40	$2^3\times U_5$
41	5×2^3	1×2^0	U_{41}	42	$2^1\times U_{21}$
43	5×2^3	3×2^0	U_{43}	44	$2^2\times U_{11}$
45	3×2^4	-3×2^0	U_{45}	46	$2^1\times U_{23}$
47	3×2^4	-1×2^0	U_{47}	48	$2^4\times U_3$
49	3×2^4	1×2^0	U_{49}	50	$2^1\times U_{25}$
51	3×2^4	3×2^0	U_{51}	52	$2^2\times U_{13}$
53	3×2^4	5×2^0	U_{53}	54	$2^1\times U_{27}$
55	7×2^3	-1×2^0	U_{55}	56	$2^3\times U_7$
57	7×2^3	1×2^0	U_{57}	58	$2^1\times U_{29}$
59	1×2^6	-5×2^0	U_{59}	60	$2^4\times U_{15}$
61	1×2^6	-3×2^0	U_{61}	62	$2^1\times U_{31}$
63	1×2^6	-1×2^0	U_{63}	64	$2^6\times U_1$
65	1×2^6	1×2^0	U_{65}	66	$2^1\times U_{33}$
67	1×2^6	3×2^0	U_{67}	68	$2^2\times U_{17}$
69	1×2^6	5×2^0	U_{69}	70	$2^1\times U_{35}$
71	1×2^6	7×2^0	U_{71}	72	$2^3\times U_9$
73	5×2^4	-7×2^0	U_{73}	74	$2^1\times U_{37}$
75	5×2^4	-5×2^0	U_{75}	76	$2^4\times U_{19}$
77	5×2^4	-3×2^0	U_{77}	78	$2^1\times U_{39}$
79	5×2^4	-1×2^0	U_{79}	80	$2^4\times U_5$
81	5×2^4	1×2^0	U_{81}	82	$2^1\times U_{41}$
83	5×2^4	3×2^0	U_{83}	84	$2^2\times U_{21}$
85	5×2^4	5×2^0	U_{85}	86	$2^1\times U_{43}$
87	5×2^4	7×2^0	U_{87}	88	$2^3\times U_{11}$
89	3×2^5	-7×2^0	U_{89}	90	$2^1\times U_{45}$
91	3×2^5	-5×2^0	U_{91}	92	$2^2\times U_{23}$
93	3×2^5	-3×2^0	U_{93}	94	$2^1\times U_{47}$
95	3×2^5	-1×2^0	U_{95}	96	$2^5\times U_3$
97	3×2^5	1×2^0	U_{97}	98	$2^1\times U_{49}$
99	3×2^5	3×2^0	U_{99}	100	$2^2\times U_{25}$
101	3×2^5	5×2^0	U_{101}	102	$2^1\times U_{51}$
103	3×2^5	7×2^0	U_{103}	104	$2^3\times U_{13}$
105	7×2^4	-7×2^0	U_{105}	106	$2^1\times U_{53}$
107	7×2^4	-5×2^0	U_{107}	108	$2^2\times U_{27}$
109	7×2^4	-3×2^0	U_{109}	110	$2^1\times U_{55}$
111	7×2^4	-1×2^0	U_{111}	112	$2^4\times U_7$
113	7×2^4	1×2^0	U_{113}	114	$2^1\times U_{57}$
115	7×2^4	3×2^0	U_{115}	116	$2^3\times U_{29}$
117	7×2^4	5×2^0	U_{117}	118	$2^1\times U_{59}$
119	7×2^4	7×2^0	U_{119}	120	$2^3\times U_{15}$
121	1×2^7	-7×2^0	U_{121}	122	$2^1\times U_{61}$
123	1×2^7	-5×2^0	U_{123}	124	$2^2\times U_{31}$
125	1×2^7	-3×2^0	U_{125}	126	$2^1\times U_{63}$
127	1×2^7	-1×2^0	U_{127}	128	$2^7\times U_1$

Our recoding is highly *redundant*, *i.e.,* each $|Q_j|$ may have several notations in Z_1 and Z_2 digits. We *fully exploited* this property to minimize the number of adders using a C-program which exhaustively explores for each odd $|Q_j|$, all possible notations and selects the least adder consumer combination according to the following priority order: $(A_j, B_j)=(A_j, 0)$; $(A_j, B_j)=(1, 1)$; $(Z_1, Z_2)=(1\times 2^7, Z_2)$; and finally $(Z_1, Z_2)=(Z_1, 1\times 2^0)$. These two latter couples allow the following simplification:

$$\ldots+\left[(1\times 2^7+Z_2)\times 2^{8j}\right]+\left[(Z_1-1\times 2^0)\times 2^{8j+8}\right]+\ldots=\ldots+\left[Z_2\times 2^{8j}-2\right]+\left[Z_1\times 2^{8j+8}\right]+\ldots$$

In case none of those cases is encountered, C-program pursues in the following priority order: $(Aj, Bj)=(1,3)$ or $(3,1)$; $(Aj, Bj)=(3, 3)$; $(Aj, Bj)=(1,5)$ or $(5,1)$; $(Aj, Bj)=(5, 5)$; $(Aj, Bj)=(1, 7)$ or $(7, 1)$; $(Aj, Bj)=(7, 7)$; $(Aj, Bj)=(3,5)$ or $(5,3)$; $(Aj, Bj)=(3,7)$ or $(7,3)$; $(Aj, Bj)=(5,7)$ or $(7,5)$. This order maximizes the occurrences of 1, then of 3, and minimizes those of 5 and 7 in $|Qj|$ digits, which will more likely reduce the number of adders in the whole C recoding. Furthermore, we perform common U_k digit elimination as an ultimate optimization step. Only odd $|Qj|$ digits are optimized. Optimized even digits are directly derived from odd ones using shift operations as indicated in Table II.

To illustrate the idea, the product $P=23453\times X$ is first computed in CSD and then in R3. It gives:

$$P_{CSD}=2^{15}\times X-2^{13}\times X-2^{10}\times X-2^7\times X+2^5\times X-2^2\times X+X;$$
$$P_{R3}=2^8\times(2^5\times U_3-2^2\times U_1)-(2^5\times U_3+U_3)\ ;\ U_1=X \text{ and } U_3=2\times X+X.$$

P_{CSD} requires 6 operations, while P_{R3} needs only 4. Note that the naïve add-and-shift algorithm would have required 9 operations. We assume that addition and subtraction have the same area/speed cost, and that shift is costless since it can be realized without any gates using hard wiring. Note that in R3 there is *no* overflow risk since the shift span is fully controlled.

III. Result Comparison

In equation (4), there are $N/8$ iterations. Each iteration generates a maximum of 2 partial products (PP). Thus, the maximal number of PP is $N/4$. A maximum of 3 supplementary adders are necessary in case 3×X, 5×X, and 7×X are all invoked at the same time in the recoding. Therefore, the maximal number of additions per coefficient (*Upb*) is bounded by $(N/4)+2$. As for the average number of additions (*Avg*), it has been exhaustively calculated for C values varying from 0 to 2^N-1, for N=8, 16, 24, and 32. But for N=64, we calculated the average using 10^5, 10^6, 10^9 and 10^{10} *uniformly distributed random C* values. While the difference between the four obtained results is insignificant ($<10^{-3}$), the average decreases as the number of C values increases, and converges to 14.4932 additions. Results are reported in Table III. For N=64, R3 uses 29.10% less additions than CSD. The saving seems to grow linearly for low values of N. It will asymptotically converge to an upper limit which is unknown for the time being.

Regarding computation-time complexity, it is *linearly* proportional to N as shown by eq. (4). As for the storage complexity, a look-up table with 128 entries is required, which is insignificant.

Concerning DBNS, Dimitrov [3] calculated average and upper-bound values from 10^5 uniformly distributed *random* constants, for 32 and 64 bits only (Table IV). Note that DBNS upper-bounds will be higher if the worst cases are not attained by the pattern of 10^5 constants.

TABLE III: R3 VERSUS CSD : AVERAGE NUMBER OF ADDITIONS (*Avg*) AND UPPER BOUND (*Upb*)

Constant Bit-width N	CSD		R3		Saving
	Avg	*Upb*	*Avg*	*Upb*	(*Avg*,%)
8	1.7882	4	1.7254	3	3.5119
16	4.4445	8	4.1050	6	7.6386
24	7.1111	12	6.2846	8	11.6226
32	9.7777	16	8.3194	10	14.9145
64	20.4444	32	14.4932*	18	29.1091

*: Obtained from 10^{10} uniformly distributed random C values.

TABLE IV: R3 VERSUS DBNS : AVERAGE NUMBER OF ADDITIONS (Avg) AND UPPER BOUND (Upb)

Constant Bit-width N	DBNS [3]		R3		Saving
	Avg	Upb	Avg	Upb	(Avg,%)
32	≈9.05+*	13*	8.3194	10	8.0729
64	16.2151*	21*	14.4932	18	10.6191

+: Taken from Fig.1 in [3]; *: Obtained from 10^5 uniformly distributed random constants.

Another performance indicator of the recoding is the smallest value that requires q additions, for q varying from 1 to the upper-bound of the recoding. Table V summarizes this information for 32-bit constant. Note that starting from q=7, higher values are provided by R3 algorithm.

TABLE V: R3 VERSUS CSD : SMALLEST VALUES FOR 32-BIT CONSTANT

Number of Additions (q)	CSD	R3
1	3	3
2	11	11
3	43	43
4	171	139
5	683	651
6	2731	2699
7	10923	34971
8	43691	559259
9	174763	17336475
10	699051	143163547
11	2796203	–
12	11184811	–
13	44739243	–
14	178956971	–
15	715827883	–

Predictability in addition-number (*Upb* and *Avg*) and runtime/storage requirements informs on the heuristic capabilities and limitations. *Upb* denotes exactly the length of the critical-path formed by successive additions, while *Avg* gives an idea on the compression performance of the heuristic. On the other hand, runtime/storage complexity helps to decide whether the use of the heuristic is appropriate with regard to a constant bit-width (N). While this latter is known for all heuristics (Table I), addition complexity is *unknown* for most of them [1][11]. Pinch was the first to set an asymptotic

complexity $O(N/\log(N))$ for *Upb* [18]. Better, based on DBNS arithmetic [19], Dimitrov [20] gave a rough evaluation of the hidden constant (α) in the big *O*-notation as being $1 \leq \alpha \leq 2$. Only CSD and R3 do have exact analytic expressions for addition complexity (only *Upb* for R3). For the all remaining heuristics, *no* addition complexity does exist. This is a real handicap as there is no visibility on how the heuristic evolves with respect to *N*, unless to exhaustively calculate *Avg* (Fig. 2) and *Upb*, but this is still limited to low values of ($N \leq 32$) as an excessive compute power is required. Though heuristics of Fig. 2 exhibits higher compression ratios than R3 for $N>16$, some values of Table VI are not only greater than the ones provided by R3, but also equal or even *greater* than *Upb* of R3. For $N \geq 128$, only Lefèvre algorithm remains practical $O(N^3)$, because even when neglecting the hidden constant α in $O(N^6)$, Hcub requires more than 4398 billions of iterations. Another serious drawback of non-recoding heuristics is the *overflow risk* because of uncontrolled shift spans [3]. Such a problem *never* occurs in digit-recoding heuristics: CSD, DBNS and R3.

It becomes now clear why despite the large number of existing heuristics; CSD is not only used in designing the *vast majority* of LTI systems [10], but incorporated in most of advanced synthesis tool as well, such as in Synopsys Design Compiler Ultra [10][21].

IV. Conclusion and Future Work

An efficient alternative (R3) to the most commonly used heuristic (CSD) has been proposed. Instead CSD, the use of R3 in designing LTI systems leads to much less power consumption and more speed. A pending issue is to determine the analytic expression of the average number of additions (*Avg*) needed by R3 with regard to constant bit-width *N*.

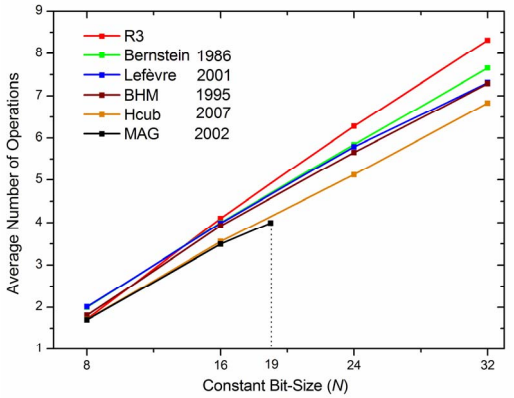


Figure 2. Comparison of R3 with non-recoding heuristics based on average number of additions (*Avg*)

TABLE VI: Number of Adders: Some Peculiarities

Algorithm	Hexadecimal Values			
	$(84AB5)_H$ N=20	$(595959)_H$ N=24	$(64AB55)_H$ N=24	$(5959595B)_H$ N=32
Bernstein [12]	8^G	7	7	8
Hcub* [6]	6	8^E	9^G	–
BHM* [13]	5	7	7	–
Lefèvre [4]	4	8^E	6	11^G
R3	4	5	6	8

*: Limited to 26 bits; x: Lowest number of additions ; *N*: Constant bit-size;
E: Equal to *Upb* of R3; G: Greater than *Upb* of R3; *Upb* of R3= $(N/4)+2$

References

[1] J. Thong and N. Nicolici, "An optimal and practical approach to single constant multiplication," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 9, pp. 1373–1386, September 2011.

[2] A. Avizienis, "Signed-digit number representation for fast parallel arithmetic,". IRE Trans. on Electronic Computers, vol. EC-10, No. 3, pp. 389–400, September 1961.

[3] V.S. Dimitrov, L. Imbert, and A. Zakaluzny, "Multiplication by a Constant is Sublinear," Proceedings of the 18th IEEE Symposium on Computer Arithmetic (ARITH'18), pp. 261-268, June 2007.

[4] V. Lefèvre, "Multiplication by an Integer Constant," INRIA Research Report, No. 4192, Lyon, France, May 2001.

[5] N. Boullis and A. Tisserand, "Some Optimizations of Hardware Multiplication by Constant Matrices," IEEE Trans. on Computers (TC), vol. 54, No. 10, pp. 1271-1282, October 2005.

[6] Y. Voronenko and M. Püschel, "Multiplierless Multiple Constant Multiplication," ACM Trans. on Algorithms (TALG), vol. 3, No. 2, Artcle 11, pp. 1-38, May 2007.

[7] A. Dempster and M. Macleod, "Using Signed-Digit Representations to Design Single Integer Multipliers Using Subexpression Elimination," Proceedings of the IEEE International Symp. on Circuits and Systems (ISCAS), vol. 3, pp. III-165–168, Vancouver, Canada, May 2004.

[8] O. Gustafsson, A.G. Dempster, and L. Wanhammar, "Extended Results for Minimum-Adder Constant Integer Multipliers," Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), vol. 1, pp. I-73 I-76, Scottsdale Arizona, USA, May 2002.

[9] F. de Dinechin, "Multiplication by Rational Constant," IEEE Trans. on Circuits and Systems II: Express Brief, vol. 59, No. 2, pp. 98–102, February 2012.

[10] R. Kastner, A. Hosangadi, and F. Fallah, "Arithmetic Optimization Techniques for Hardware and Software Design," Cambridge University Press, ISBN-13 978-0-521-88099-2, © 2010.

[11] O. Gustafsson, "Lower Bounds for Constant Multiplication Problems," IEEE Trans. on Circuits and Systems II: Express Brief, vol. 54, No. 11, pp. 974–978, November 2007.

[12] R.L. Bernstein, "Multiplication by Integer Constant," Software – Practice and Experience 16, 7, pp. 641-652, 1986.

[13] A.G. Dempster and M.D. Macleod, "Use of Minimum Adder Multiplier Blocks in FIR Digital Filters," IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing 42, 9, pp. 569-567, 1995.

[14] S. Homayoon and A. Gupta, "A Generalized Multibit Recoding of Two's Complement Binary Numbers and its Proof with Application in Multiplier Implementation," IEEE Trans. on Computers (TC), vol. 39, N° 8, August 1990.

[15] P.M. Seidel, L. D. McFearin, and D.W. Matula, "Secondary Radix Recodings for Higher Radix Multipliers," IEEE Trans. on Computers (TC), vol. 54, N°2, February 2005.

[16] A.K. Oudjida, N. Chaillet, A. Liacha, and M.L. Berrandjia "New High-Speed and Low-Power Radix-2^r Multiplication Algorithms," Proceedings of the 11th edition of IEEE-FTFC Low-Voltage Low-Power Conference, ISSN: 978-1-4673-0821-2/12, Paris, June 2012.

[17] A.K. Oudjida, N. Chaillet, A. Liacha, and M.L. Berrandjia, "A New Recursive Multibit Recoding Algorithm for High-Speed and Low-Power Multiplier," Journal of Low Power Electronics (JOLPE), vol. 8, N° 5, pp. 579-594, ISSN 1546-1998, American Scientific Publishers (ASP), December 2012.

[18] R. G. E. Pinch, "Asymptotic Upper Bound for Multiplier Design," Electronics Letters, vol. 32, N° 5, pp. 420-421, February 1996.

[19] V.S. Dimitrov, G.A. Jullien, and W.C. Miller, "Theory and Applications of the Double-Base Number System," IEEE Trans. on Computers (TC), vol. 48, No. 10, pp. 1098-1106, October 1999.

[20] V.S. Dimitrov, K.U. Järvinen, and J. adikari, "Area Efficient Multipliers Based on Multiple-Radix Representations," IEEE Trans. on Computers (TC), vol. 60, N° 2, pp 189-201, February 2011.

[21] Synopsys Datasheet, Design Compiler Ultra-Design Compiler® at its Best. Available at: www.synopsys.com.